NISTIR 8318

DADS: The On-Line Dictionary of Algorithms and Data Structures

Paul E. Black

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8318

National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8318

DADS: The On-Line Dictionary of Algorithms and Data Structures

Paul E. Black
Software and Systems Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8318

September 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8318
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8318, 40 pages (September 2020)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8318

Abstract

The Dictionary of Algorithms and Data Structures (DADS) is a publicly accessible dictionary of generally useful algorithms, data structures, algorithmic techniques, archetypal problems, and related definitions available at https://nist.gov/DADS/. DADS is meant to be a resource for the practicing programmer, although students and researchers may find it a useful starting point. DADS has fundamental entries in areas such as theory, cryptography and compression, graphs, trees, and searching, for instance, `Ackermann's function`, `quick sort`, `traveling salesman`, `big O notation`, `merge sort`, `hash table`, and `Byzantine generals`.

This document is a manual for and documentation of DADS. It records the history of DADS and explains the theory, goals, and operating philosophy behind DADS, the structure of the web site, and the structure and content of the DADS source directory. This report also describes in detail the format and content of *term files*, which correspond roughly to entries, and the software to compile term files into Hyper Text Markup Language (HTML) web pages and to check and maintain DADS. Finally, there are notes on possible future enhancements. A typical term is included as an appendix for the reader's convenience.

Key words

Algorithm; data structure.

Table of Contents

1. Introduction

1.1 What is DADS?

The Dictionary of Algorithms and Data Structures, or DADS https://nist.gov/DADS/, is a publicly accessible dictionary of generally useful algorithms, algorithmic techniques, data structures, archetypal problems, and related definitions. Algorithms include common functions, such as `Ackermann's function`. Problems include `traveling salesman` and `Byzantine generals`. Many entries have links to implementations and to more information. DADS includes index pages that list entries by type and area, such as theory, cryptography and compression, graphs, trees, and searching. It has features to make it easier to use for more people, like a two-level index for users with limited-bandwidth; the two-level index requires 1/20 the download as the regular pages.

DADS was created and is maintained and made available by National Institute of Standards and Technology (NIST), Information Technology Laboratory, Software Quality Group.

This report documents the history, present structure, and possible future extensions of DADS. Section 2 explains the present theory, goals, and operating philosophy behind DADS and the structure of the web site. Section 3 describes the format and content of *term files*, which correspond roughly to entries in the dictionary. Section 4 covers the structure and content of the DADS directory and software and special files to generate HTML files, edit or rename term files, check for errors, and maintain DADS. Finally, Sec. 5 contains notes on future enhancements. Appendix A is a typical term file included for the reader's convenient reference.

1.2 Typographical Conventions

A DADS entry is in monospaced font: `acyclic graph`. Text that is the raw source of DADS is also in monospaced font in a separate paragraph, not justified or filled, just as it appears in a text editor:

```
@DEFN=A wonderful wife.
Every man should have such an incredible wife.
```

Computer commands are also shown in the above monospaced separate paragraph.

As usual, double quotes indicate the use of the word itself, not its semantic meaning. For example, the plural of "vertex" is either "vertices" or "vertexes."

1.3 History

DADS began in 1998 when Phillip Laplante issued a call for editors in various areas for a new book, Dictionary of Computer Science, Engineering and Technology, to be published by CRC Press. In September we volunteered to be the editor for the area of algorithms and data structures. We created a web site in order to get corrections and additions from the

community and to share whatever work had been done to that time. We chose "Dictionary of Algorithms And Data Structures" for its short but memorable acronym: DADS.

Within a few days we wrote 47 entries. In November we added another 53 entries extracted from Gaede and Günther's "Multidimensional Access Methods" [1], then added another 48 in December. We realized the web site would only attract a lot of visits, to lead visitors to offer corrections and suggestions, if it had a lot of content, so we continued adding entries. Big increases came in February 1999 (293 entries) and June (290). 265 of the June additions came by permission from CRC Press's "Algorithms and Theory of Computation Handbook." Laplante was the editor of that book, too, and suggested getting definitions from it. Nancy Laplante manually rekeyed the entries from the printed book to create plain text, which we turned into entries. By the end of 2000, DADS had 993 entries. As of May 2020 there were 1301 entries in the main index and 1592 web pages, which includes pages for aliases and web-only names and index pages.

DADS has benefited over the years from 31 different authors of entries and many contributors of notes. DADS has historical notes from Marlene Metzner Norton and Jack E. Bresenham and incorporates communication from Donald E. Knuth.

DADS has thousands of accesses from all over the world. In January 2012, the site had over 28 000 unique visitors who downloaded some 80 000 pages. Visitors come from (from most frequent to least) .net, .com, .edu, .in, .de, .gov, .uk, .pl, .fi, and others. Most visits are short (84 % less than 30 s) with a few hundred going two minutes or five minutes or more. The most frequently accessed entries were `big O notation`, `cube root`, `manhattan distance`, `euclidian distance`, `square root`, and `sparse matrix`. In addition the preceding, other top search key phrases or words are `algorithms`, `binary search`, `algorithm`, `adjacency list`, `tree`, `structure`, and `notation`.

2. DADS Today

2.1 Theory and Goals

2.1.1 Entries

> A dictionary definition draws some important distinctions and serves to remind the reader of something that is presumed to be familiar already; it is not intended as a catalogue of general knowledge. There is a place for encyclopedias as well as dictionaries. [2]

Entries in the Dictionary of Algorithms and Data Structures are meant to be short. The brevity helps the web pages rank high in Google and other web searches and helps users use DADS on small-screen devices. The goal is for definitions to be less than 100 words, if possible. Complex algorithms are just outlined. For example, just the major phases or steps are given. Most entries include notes, links to explanations, and other information, which were not part of the printed dictionary,

The target audience of DADS is the working programmer, hence it includes links to implementations which we found throughout the web. To help students and academicians,

DADS includes bibliographic citation of first published references and gives credit to inventors, for instance, the Fisher-Yates shuffle.

2.1.2 What is Included or Not?

DADS is meant to be a general-purpose or basic reference, not a complete record of every algorithm or data structure invented. In particular it does not include business data processing, communications and networking, operating systems or distributed algorithms, programming languages, artificial intelligence, graphics, or numerical analysis. Each field requires (and would deserve!) a complete reference by itself.

We exclude unpublished work. Not infrequently we receive newly invented data structures or algorithms. Except for a few simple sorting algorithms, we don't have the resources to check new work and decline to appear to endorse such work.

We added some entries for personal use, such as the `Byzantine generals` problem and `mean`, `median`, and `mode`, because we had trouble remembering them and couldn't find a good, convenient reference.

DADS uses American spelling, not British spelling. For instance, color instead of colour and traveling instead of travelling. See Sec. 3.2 for additional style specifics. Spelling or naming variants of an entry may be included using the @AKA (Also Known As) or @WEB fields, described in Sec. 3.3.6.

There are bits of whimsy scattered throughout DADS, for instance, in the entries for `brute force`, `random number generator`, `depth-first search`, `NP-complete`, `trie`, `selectionsort`, and `Marlena`.

2.1.3 The Web Site

The web site has a main page with an index having links to the entries. The main page also links to the following automatically-generated, additional web pages.

- An index by area, such as graphs, searching, sorting, theory, and trees,
- An index by type, such as data structures, algorithms, and definitions,
- Entries that have links to implementations, and
- *All* entries, including spelling and name variants. This supplemental page exists so web-crawling robots find entries that are meant only for web searches, such as `travelling salesman` (the British spelling with two "l"s), `quick sort` (in addition to `quicksort`), and misspellings of `algorithm`.

Links to an authors page and to a page giving citation examples and explanation of credits are also included on the main page. Finally it includes links to references on algorithms and data structures and a bibliography we have drawn entries from.

To serve users with limited bandwidth, there is a minimal two-level interface, which is at https://www.nist.gov/dads/terms2.html. The first page is a very sparse page with little more than a list of second-level index pages. The number of second-level pages and

their positioning in the directory structure was designed to minimize the number of bytes needed to reach any page. Access through the two-level interface requires about 1/20 the number of bytes as using the main index and web pages. More details about the design and requirements of the minimal interface are in Sec. 4.3.1.

To serve a broad community and reduce maintenance burdens, DADS is a "low tech" web site. Instead of hypertext markup language (HTML) 5, complex javascripts, or even dynamic pages, it consists of static pages generated from term files by a program, described in Sec. 4.3. The goal is for the pages to be usable by as many browsers as possible—including text-only browsers like Lynx—and accessible by individuals with visual disabilities. Entries include figures when helpful, but it is not meant to be entertainment.

2.2 Serve the Community

The Dictionary has the opportunity to contribute to society in many ways.

The Dictionary can reconcile usage and definition. For example `height` is defined differently by different authors. (We chose one that was consistent with other definitions and led to clear, simple formulations and definitions.) As another example, in October 2002 we corresponded with Mildrid Ljosland, who was writing a book, and found inconsistent definitions of "full binary tree" in literature. After reviewing many authors, we chose definitions for "full," "perfect," and "complete." After having defined three types of trees, we explained our reasoning:

> Type 3 (a binary tree for a heap implemented in an array) is called "complete" by everyone that I've looked at and even CLR hints at this definition. So I'd call this "complete."
>
> Most authors that mention it use "full" to refer to Type 1 (all nodes have either 0 or 2 children).
>
> It is exactly on type 2 where there is disagreement. Since there is consensus on the other two types (and one uses the name "full"), I'd adopt Priess' term of "perfect," for something with 2^h - 1 nodes. [3]

Entries capture ambiguous definitions, such as in `binary tree` above. They also display different definitions with the same name. For example the definition of `degree` is

> (1) Of a vertex, the number of edges connected to it. (2) Of a graph, the maximum degree of any vertex. (3) Of a tree node, the number of child nodes it has.

In other words, the degree of a vertex is the number of edges connected to it. The degree of a graph is the maximum degree of any vertex. The degree of a tree node is the number of child nodes it has.

Another example is that the `tree` data structure has subtle but important differences compared to a `tree` graph.

Entries include spelling and naming differences. For instance doubly-linked list is also known as (@AKA) two-way linked list and symmetrically linked list. Also, bubble sort is known as sinking sort and as exchange sort. In cases where a different name would appear very close to the original in the main index, we use the @WEB field, which only appear on the page with *all* entries. An example of a @WEB field is Ackermann's function, which is sometimes referred to as the Ackermann-Peter function. See Sec. 3.3.6 for more detail.

DADS has several structured ways to connect entries. Some data structures are variants or specializations of others. For example, binary tree is a kind of or specialization of tree and k-ary tree (kept in the @IMA for "I'm a ..." filed) and its variants (in the @VARIANT field) are complete binary tree, full binary tree, binary search tree, binary heap, balanced binary tree, threaded tree, Fibonacci tree, and extended binary tree.

Some concepts are a part of or used in other concepts. This is indicated by @IMIN (I'm in ...) and the reciprocal relation, which is @INME ([this is] in me). As examples, edge is in graph, Euler cycle is in Christofides algorithm, depth is a concept in tree, Gray code is used in Karnaugh maps, separate chaining is a concept in hash tables, and divide and conquer is a technique used in heapify, quicksort, binary search, and other algorithms.

We compare and contrast entries where helpful. For instance, Fisher-Yates shuffle may be seen as a selection sort running in reverse. We contrast an Euler cycle with a Hamiltonian cycle and a Monte Carlo algorithm with a Las Vegas algorithm as notes in both entries. We give examples for mean, median, and mode.

Three dozen DADS entries have mathematically formal definitions. These formal definitions allow formal reasoning, such as proofs of correctness. For instance, we give axiomatic semantics definitions for several entries and mathematical definitions for others. The term queue has the following formal definition in axiomatic semantics:

> new() returns a queue front(add(v, new())) = v remove(add(v, new())) = new() front(add(v, add(w, Q))) = front(add(w, Q)) remove(add(v, add(w, Q))) = add(v, remove(add(w, Q)))

From these axioms, the correct result of any sequence of queue operations can be deduced without resort to a specific implementation.

The Dictionary includes other tidbits, like the pronunciation of trie, deque, Euler, and Karnaugh. The Dictionary also has more than a dozen historical notes we found in researching, such as, the background of Dutch National Flag algorithm, and personal communications from Marlene Metzner Norton about the incorrect naming of shell sort, from Jack E. Bresenham about his line drawing algorithm, and from Alan J. Goldman about the naming of the chinese postman problem.

Finally most entries have "See also" (@XREFS, see Sec. 3.4.1) entries for concepts with other relations, like an alternative algorithm, a similar data structure, or a related concept.

3. Entries

Each term file corresponds roughly to one entry in the on-line dictionary. A term file may result in additional entries for web-only or alternate names, see Sec. 3.3.6. Term files are turned into entries and web pages by a program, mkterms. Since there are so many different, subtle tasks in that process, we occasionally refer to it as "compiling terms." See Sec. 4.3 for more details of the operation of mkterms and Sec. 5.2.1 for thoughts on future directions.

3.1 How Complete Should An Entry Be?

How much should go into an entry? Our suggestion is to not try to include everything possible for the very first version of an entry. Start with a definition and enrich it with other information later. One way to enrich an entry is to think of and add related entries. Section 3.3.7 suggests different kinds of relations to look for.

Another way to enrich entries, and make the Dictionary a more coherent whole, is to search the Dictionary. Is the entry already defined under a different name? Does it overlap with, contradict, or generalize other entries? You may find that several entries need to be updated or created on the basis of one submission.

It further enriches the Dictionary to include links to extended explanations, sites with source code implementations, or citations of an original paper defining it.

3.2 General Style in the Term File

This section documents a general style, which we try to follow for consistency. We adapted it from the Free On-Line Dictionary of Computing (FOLDOC) style [4].

- Use American spelling except where names or titles use British: "-ize" rather than "-ise," "color," "gray," "behavior," "humor," etc.
- Acronyms should be in the @AKA or @WEB fields (Sec. 3.3.6) of the term, instead of creating a separate term file. They become just links to the main entry. For example, `ADT` or `DFS`.
- Write acronyms without "."s, e.g., "SMTP," but write Latin acronyms as "e.g.," "i.e.," etc.
- Separate paragraphs by a single blank line with no initial indentation.
- Write decades and pluralized acronyms without apostrophes, e.g., "1960s," "GUIs."
- Write numbers zero to ten as words, 11 and up as digits, 10 000 and up with spaces, not commas, e.g., not 10,000.
- Hyphenate number-unit pairs used as adjectives. e.g., "a 32-bit bus," "a one-megabyte address space." Otherwise the number and unit should be separated by one space.
- Never use "utilise" (or "utilize"), use "use."
- Put a comma before "and" or "or": e.g., "A, B, and C."

Use plain HTML for paragraph breaks (for example, in `sort`), lists (`dictionary`), tables (`sieve`, `upper triangular matrix`), text pictures (`optimal merge`, `square root`), and formatting source code (`tail recursion`).

3.3 Elements of the Term File

To make it easier to enter and process, information is entered in fields or sections, like a semantic markup. A field starts with an at-sign (@) at the beginning of a line, the field name, and an equal sign (=):

```
@NAME=inverted index
# the definition
@DEFN=An index into a set of {texts} of the words in the texts.
The index is accessed by some {search} method.  Each index entry
gives the
word and a list of texts, possibly with locations within the
text, where the word occurs.
```

The field continues across lines until the next field or the end of file. A comment line begins with a pound sign (#) and is ignored.

To write a new entry, copy the empty template term file, `template.trm`, and fill it in. It includes notes to remind the author of the contents of each field.

Below is each field. The order of fields does not matter; they are compiled into their proper places. We present the fields here in what we hope is a logical order.

3.3.1 @NAME

The name of the term. LaTeX-like expressions, as detailed in Sec. 3.4.2, are valid here.

```
@NAME=$B<sub>k</sub>$ tree
@NAME=$\Theta$
@NAME=partial recursive function
```

The name should appear with the case and punctuation it would have in the middle of a normal sentence: all lower-case if that's how it's normally written.

3.3.2 @TYPE

A letter indicating the general classification. These are defined in the file `types.data`.

A Algorithms
T Algorithmic Techniques
D Definitions
P Classic Problems
S Data Structures

This has no impact on the structure of the Dictionary, but seems to be useful.

3.3.3 @AREA

This term's general area. We added this field to get better coverage. That is, we would examine all the "graph" entries, for instance, to see if any were missing. This field may no longer be useful, and there may be overlap. These are defined in `areas.data`.

basic Basic; very fundamental notations; like "point" and "line" in geometry.
autom Automata and State Machines
theory Theory
graph Graphs
tree Trees
search Searching
sort Sorting
crypt Cryptography and Compression
extern External Memory Algorithms and Data Structures
para Parallel
verf Verification and Formal Methods
geom Computational Geometry
combin Combinatorics
numeric Numeric Computation
quant Quantum Computation

3.3.4 @DEFN

Write the definition as if there is a lead-in phrase "A (whatever) is ..." For instance, for abstract data structure, don't write "An abstract data structure is a set of data values ..." Just write "A set of data values ..." HTML, cross references (Sec. 3.4.1), and LaTeX-like mark-up (Sec. 3.4.2) are valid.

If a single entry has multiple meanings, give each meaning a single paragraph tagged by a parenthesized decimal integer incrementing from 1, for instance,

```
@NAME=tree
# _A_lgorithm, _D_efinition, _P_roblem, or data _S_tructure
@TYPE=S
@AREA=tree
@DEFN=(1) A data structure accessed beginning at the {root} node.
Each
{node} is either a {leaf} or an {internal node}.  An internal node
has one or more {child} nodes and is called
the {parent} of its child nodes.  All children of the same node are
{siblings}.
Contrary to a physical tree, the root is usually depicted at the top
```

```
of the structure, and the leaves are depicted at the bottom.
(2) A {connected#connected graph},
{undirected#undirected graph}, {acyclic graph}.  It is
{rooted#rooted tree} and {ordered#ordered tree} unless otherwise
specified.
<br />
<img src="../Images/tree.png" height="219" width="318">
<br />
Thanks to Joshua O'Madadhain (jmadden@ics.uci.edu) for the figure, 6
October 2005.
```

Order multiple meanings from most common to most obscure.

3.3.5 @FORML

The formal definition. Only 37 entries have these, for instance, `dictionary`, `stack`, `Fibonacci number`, `total order`, and `tree`. We include these if succinct and helpful. Maybe someday most entries will have formal definitions that can be automatically processed, say in a theorem proving system.

3.3.6 @AKA and @WEB

Aliases for this entry, or, names that it is Also Known As. Any alias is added to the main index (and links to original page). A separate web page is created for each aliases, which refers to the original page. Here are some examples of @AKA names:

```
bidirectional bubble sort: cocktail shaker sort, shaker sort, double-direction bubble sort
depth first search: DFS
deque: doubly-ended queue
Euler cycle: Eulerian path, K&ouml;nigsberg bridges problem
inplace sort: sort in place
inverse Ackermann function: $\alpha$
```

Which should be the @NAME, and which should be an @AKA? The most common is the @NAME, and other, lesser-used variants are @AKA entries.

Names should be in the @WEB field if they are uncommon or are spelled similarly, that is, if the alias would appear close in the main index. For example, `travelling salesman`, the British spelling, would be next to the main entry, `traveling salesman`, in the index, so it is @WEB.

The @WEB field has other cross-listings solely for web indexing, such as minor word or spelling variants. Web entries do not appear in the main index, but a page is created that refers to the original.

```
octtree: octree
```

Euler cycle: Euler tour, Koenigsberg bridges "Koenigsberg bridges" is @WEB since "Königsberg bridges problem" is @AKA

quicksort: quick sort

traveling salesman: travelling salesman, traveling salesman problem

The Dictionary also includes almost a dozen misspellings of `algorithm`.

3.3.7 Related Entries:@IMA, @VARIANT, @IMIN, @INME, and @XREFS

The following five fields are all comma-separated lists of cross references, which are enclosed in curly brackets ({...}). See Sec. 3.4.1 for details. Order the entries beginning with the most closely related or relevant or most likely to be of interest.

These may be near-synonyms or antonyms (Euler cycle vs. Hamiltonian cycle and sink vs. source), entries related to each other (e.g. reflexive, transitive, symmetric), aggregates where it is used (IN-ME) or that use it (I'M-IN Dijkstra's algorithm and Johnson's algorithm), significant properties (greedy algorithm), generalizations (I'M-A) or variants, and so forth.

@IMA
Generalization: "I am a kind of ..."

@VARIANT
Specialization: "... is a kind of me."

@IMIN
Aggregate parent: "I am a part of or used in ..."

@INME
Aggregate child: "... is a part of or used in me."

@XREFS
Other cross references that don't fit any of the above specific categories. Displayed as "See also ..."

Try to work cross-references into the definition or notes rather than just including them in @XREFS. This makes their relationship to the entry clearer.

Here are the cross references for `function`:

```
@IMA={relation}
@VARIANT={boolean function}, {constant function},
{unary function}, {binary function}, {trinary function},
{n-ary function}, {total function}
@IMIN=
@INME={signature}
@XREFS={procedure}, {predicate}
```

Here are the cross references for `graph`.

```
@IMA=
@VARIANT={directed graph}, {undirected graph}, {acyclic graph},
```

```
{directed acyclic graph},
{planar graph}, {connected graph}, {biconnected graph},
{bipartite graph}, {complete graph}, {dense graph}, {sparse graph},
{hypergraph}, {multigraph}, {labeled graph},
{weighted graph},
{tree}
@IMIN=
@INME={vertex}, {edge},
Implementations:
{adjacency-list representation}, {adjacency-matrix representation}
@XREFS=Relations between vertices: {adjacent}, {self-loop},
Relations between graphs: {isomorphic}, {homeomorphic}, {dual},
{subgraph},
Properties: {diameter}, {degree},
Other: {graph drawing}, {graph partition}
```

The idea to refine "See also" (@XREFS) into Generalization (@IMA), Specialization (@VARIANT), Aggregate Parents (@IMIN), and Aggregate Children (@INME) came from the Georgia Tech College of Computing Plexicon: https://cc.gatech.edu/projects/plexicon/.

3.3.8 @IMPL

The keyword suggests "implementation". HTML links to implementations with source code. The programming language is in parentheses. Put the best references first. From quicksort:

```
@IMPL=<a href="http://www.cs.princeton.edu/~rs/">Robert
Sedgewick's</a> talk showing that with Bentley-McIlroy 3-way
partitioning <a
href="http://www.cs.princeton.edu/~rs/talks/QuicksortIsOptimal.pdf">
Quicksort Is Optimal (C)</a> (pdf format) with discussion and proof.
<a
href="http://java.sun.com/applets/jdk/1.0/demo/SortDemo/example1.html">
animation and code
(Java)</a>;
```

@IMPL links are listed in a separate page so programmers can find code quickly.

Won't this help students cheat on programming assignments? Yes, it might. But we hope the benefit to legitimate users, such as working programmers, outweighs the problem. Having taught programming classes, we believe that tests and other assignments will tend to catch a cheater.

3.3.9 @LINKS

HTML links to additional web sites with explanations, diagrams, animations, etc. If it has source code, list it in @IMPL. From `busy beaver`:

```
@LINKS=History, explanation, and links about the
<a href="http://grail.cba.csuohio.edu/~somos/bb.html">Busy Beaver
Turing Machine</a>.
Heiner Marxen's
<a href="http://www.drb.insel.de/~heiner/BB/">currently known busy
beaver results</a> page.
```

We leave notes within comments in the term file to help find the page again when (not "if") it moves, for instance, author's name, page name, or institution.

3.3.10 @BIB

One or more bibliographical references, for instance, to the original defining article or to an early definitive article. These are pure HTML. Information is author (full name), title, publication (e.g., conference or journal), volume(number):pages, and date. Multiple references are separated with . That is, separate paragraphs by ending one with then starting the next one with . These might seem mismatched, but mkterms adds before everything and after everything. Here is `Ackermann's function` (we divide long lines to fit):

```
@BIB=<strong>Wilhelm Ackermann</strong>,
<em>Zum Hilbertschen Aufbau der reellen Zahlen</em>,
Mathematische Annalen 99(1):118-133, December 1928.<br />
<a href="http://dx.doi.org/10.1007/BF01459088"
                            target="_blank">doi:10.1007/BF01459088</a>
</p>

<p>
The formal definition given here is G<sub>n</sub>x in the first
                                                        page of<br />
<strong>Raphael M. Robinson</strong>,
<em>Recursion and Double Recursion</em>,
Bulletin of the American Mathematical Society 54:987-993, October
                                                          1948.<br />
<a href="http://dx.doi.org/10.1090/S0002-9904-1948-09121-2"
                    target="_blank">10.1090/S0002-9904-1948-09121-2</a>
```

3.3.11 @HISTORY

Historical notes. We collected a dozen first-person or historical notes from computing pioneers, and we want to preserve them. Entries with this field are `Dutch National Flag problem`, `binary tree`, `chinese postman problem`, `hash table`, `shell sort`, `ideal random shuffle`, and `trie`.

3.3.12 @NOTES

Additional notes. These are secondary explanations separate from the succinct definition. These may include comparison (from `Euler cycle`: "A Hamiltonian cycle includes each vertex once; an Euler cycle includes each edge once."), pronunciation (from `Karnaugh map`: "Pronounced 'car-no'." [We called the university in New Jersey where he taught to confirm it.]), explanation (see `graph` and `square root`), bibliographical references (sources consulted to write the definition), or anything else we took time to write down or look up and want to make available.

Notes may have HTML, cross references ($\{\ldots\}$), and LaTeX-like expressions. If something has a uniform resource locator (URL), it should be in @LINKS, not @NOTES.

3.3.13 @AUTHOR

The author's initials from `authors.data`. These are hyperlinked to their entry in the contributor's page. Separate multiple initials with commas.

3.4 Automatic Cross References and Mathematical Expressions

3.4.1 Cross References

Cross references link to other entries in the Dictionary. They become hyperlinks while compiling terms. A cross reference is text enclosed in curly brackets ($\{\ldots\}$) and may cross lines. For instance, the definition of `strictly increasing` begins

```
@DEFN=A {function} from a {partially ordered#partial order} {domain} to
```

Here's the final HTML (line breaks added for readability).

```
A <a href="function.html"><em>function</em></a> from a
<a href="partialorder.html"><em>partially ordered</em></a>
<a href="domain.html"><em>domain</em></a> to
```

What about different word senses? The definition in the Dictionary is "partial order," but "partially ordered" is grammatically correct. When a different word is needed, use a pound sign (#). Text before the pound sign appears in the page, and text after it is the entry name.

The entry name, `partial order`, was replaced by the term file containing that @NAME, which is `partialorder.html`.

If a cross reference doesn't match any entry, a trailing "s" or "es" is removed or a leading upper case letter is changed to lower case to try to find the reference. The following all work.

```
the number of {nodes} in the tree
if {prefixes} of the {pattern} don't match
{Worst-case} run time is
```

Irregular plurals must be handled with the pound sign (#)

```
the {children#child} of every {node} are
```

Only cross reference the first occurrence of each entry in an article. Do not use phrases such as "qv" or "which see." Don't create chains of references; point them all at the actual entry.

Note: many wikis [5] use vertical bar (|) where we use pound sign (#). The roles are reversed, too. In Wiki it is {link|text} while we have {text#link}. In the future, we could change to the vertical bar format. We mention this in the Future section, Sec. 5.

A colon (:) separates an external source from the cross reference. For now, we only reference Wikipedia. For instance,

```
{Wikipedia:Gray code}
```

becomes (line breaks added for readability).

```
<a href="https://en.wikipedia.org/wiki/Gray code"
target="_blank"><em>Gray code [Wikipedia]</em></a>
```

An example of different words is

```
{Office#Wikipedia:National_Institute_of_Standards_and_Technology}
```

3.4.2 LaTeX-like Expressions

Mathematical expressions may be written in a form like LaTeX. Expressions begin and end with dollar signs ($...$) and may cross lines. For example

```
is sparse if $k \ll n \times m$
```

Some special symbols, like \log or \max, become text. Others, like \ll, become HTML (<<). Still others become tiny pictures, like \oplus ($\oplus$).

The biggest difference between the DADS language and LaTeX is that superscripts and subscripts have start and end marks, like HTML.

```
For two elements $e<sub>i</sub>$and $e<sub>j</sub>$
Complexity is {O(n<sup>2</sup>)#big-O notation}
```

You may put cross references inside LaTeX expressions or LaTeX expressions inside cross references.

```
time complexity {$O(E + V \log V)$#big-O notation}
```

This ad-hoc language should be replaced with a shared language, as explained in Sec. 5.4.2.

3.5 Other Entry Information

There are two special test term files. `aaaaTest.trm`, has strange, but allowed, cases and uses. `zzzzTest.trm`, has every error possible. These are described in Sec. 4.4.1.

For decades we've marked documents and programs with the creation date and the latest modified date. Our text editor, emacs, has a tiny script to update the latest modified date every time it is saved, so it doesn't take any human effort.

```
# *created  "Tue Mar 16 16:31:33 1999" *by "Paul E. Black"
# *modified "Thu Jan 10 15:39:17 2002" *by "Paul E. Black"
```

We include the creation date for history. It is sometimes surprising how long programs or documents have been around. The modified date is handy in case the file is copied: it is the date last saved, not the date the file was created or changed.

We also end all files with

```
# end $Source$
```

to be assured that it hasn't been accidentally truncated by the file system, net transmission, or some other failure. $Source$ is a keyword for the revision control system (RCS) version control system, although we no longer use RCS.

4. The Software and the Directory Structure

This section describes and documents the particular files used for DADS and its directory structure. This section also describes and documents the software used with DADS.

4.1 Files and Directories

The DADS directory structure does not have many layers, but the software does look for some item in particular directories. The main DADS directory has many required or helpful files. Many will be discussed in connection with their use. Scripts and other software, which are in the main directory, are documented later in this section. Three files, `areas.data`, `authors.data`, and `types.data`, list allowed values for entries. Other files in the main directory are

README brief introductions to files and directories

Makefile commands, scripts, and dependencies to (re)build the DADS web site and perform other operations, like spell checking. Used by the Unix "make" command. Various operations are documented later in this section.

latex2html.data conversions from DADS' LaTeX-like commands to HTML representations. See Sec. 3.4.2. Used by mkterms.

html2latex.data conversions from DADS' commands to LaTeX for the printed dictionary.

htmlWarnings HTML warnings known to be false. It is easier to keep these relatively few exceptions than perfect the software or pick out new, supposedly real, warnings. Used in Makefile scripts.

dads.spell spell check warnings known to be false. Used in Makefile scripts.

reference.checklink warnings produced by a link checker known to be false.

template.trm copy this to start a new entry. It has all fields and includes some comments.

zzzzTest.trm a test term file with invalid conditions and inconsistencies. Processing this ensures that the built-in checks are working.

aaaaTest.trm a test term file with every valid condition for complete coverage of the processing software. Documented in Sec. 4.4.1.

As a convention, directory names begin with a capital letter. Four subdirectories, Pages, Terms, Images, and Target, are used by the software to (re)build the web site. Other subdirectories are for manual use. Important subdirectories and files are stored in the git version control system.

Pages introductions, conclusions, and other pieces for various indexes and files.

Terms the term files. Each term file gives rise to one or more entries.

Target the web-ready HTML files are placed here. Its structure and content mirror web directories. At one time this was a link to a remote directory on the web server.

Images figures and pictures. It includes editable (.fig) files for some images, acknowledgments or sources, and an HTML file (gifs.html) to display all images in one page. The latter is solely for the maintainer's benefit.

UnusedImages figures and pictures that are no longer used. Most of them were gifs for math symbols before HTML included them.

aaaaTestDir expected results of compiling aaaaTest.trm. Used by aaaaTestScript.

Undefined terms that we deleted because they had no definition.

MovedToWikipedia terms that we deleted because we moved them to Wikipedia.

Deleted other terms that we deleted, for instance because they didn't fit the goals of the Dictionary.

.git versions of files are kept here. Used by git commands.

Sources some notes on the source of entries, definitions, and images. Also has permission for some material.

FutureEntries notes, tentative term files, and materials for future additions.

bin three scripts that may have some value. ang2curly converted a former angle-bracket cross references to current curly-bracket style. getCRCs reports terms that are copyright CRC. mostWanted examines http logs to report how often each entry is accessed.

Ontology various files and scripts for making DADS into an ontology.

OldChecklinks previous outputs of checklink runs. We kept them to build automatic recognition of when a link was moved or relocated and hasn't failed (error 404).

The names of many of the files and subdirectories are used in mkcommon.pl, which is described briefly in Sec. 4.3.

4.2 General Notes on Software

We try to write programs to be well-structured and commented. We also write them to be defensive or resilient since it may be months (or years) before we use something again—we'd rather embed knowledge in the code than have to remember it. For example, when we discover an error that was not caught by tests, like duplicate entry names or malformed entries, we add code to catch and report that error in the future.

We have been refactoring the code and the web pages to minimize the numbers of places that need to be changed for anything. The Unix m4 macro processes helps with web pages. Unfortunately it makes the structure complex. Users only need to understand m4 to make extensive changes to introduction and conclusion templates, explained below.

A comment in code beginning with "SKIMP" indicates that we know it should be written better, but it was not important enough at the time. For instance, the header for the implementations section is always "Implementation" (singular). It would be nicer if the plural is used, "Implementations," when there is more than one. Here's the comment and the code:

```
# SKIMP write plural if more than one
print TERMPAGE "<h2>Implementation</h2>\n$eimpls\n";
```

All the DADS code runs in a general Unix environment. For instance, it uses perl, make, m4, spell, git, cp, and diff.

Many of the final HTML files have fixed preambles or closings with generated bodies. For ease of programming and maintenance, we often create separate preamble files, with an extension of .intro (introduction), and closing files, with an extension of .concl (conclusion). Many programs generate the body in a temporary file, then concatenate the introduction, body, and conclusion to make a complete HTML page.

4.3 Compiling Terms into Web Pages

The heart of DADS is the more than one thousand individual files for entries written in plain text contained in annotating fields. These are "compiled" into web pages by a program, mkterms, which is written in perl. In summary, the program

- turns cross references into HTML href anchors,
- formats LaTeX-like mathematical expressions,
- adds headers (intros) and trailers (concls), while replacing macro variables, and does other HTML formatting,

- creates indexes, and
- creates alias (@AKA) and web-only (@WEB) pages.

The program also checks consistency and reports errors. The major steps are

1. Read configuration stuff. (Step I in the code comments)
2. Read terms into arrays, "compiling" names for them. Add separate entries for names in @AKA and @WEB fields. (Step II)
3. Start the index files: main, two-level, unified. (Step III)
4. Compile each entry into its own web page, adding entries to the index files (Step IV)
5. Finish the index files (Step V)
6. Write additional index pages: entries with implementations, entries by area, and entries by type (Steps VI, VII, and VIII)

To regenerate the site, use the "make" command. In the DADS directory enter

```
$ make site
```

This rebuilds any needed files. It then invokes mkterms to rebuild any needed terms and all the indexes. Between make/Makefile and mkterms, only the things that need it are rebuilt or recompiled.

The Makefile may invoke one auxiliary program, mkauthors, to (re)create the contrib.html page from fixed introduction and conclusion pieces and the authors.data file. It sorts the authors and formats the entries.

mkterms uses an auxiliary program, mkcommon.pl. mkterms uses most of the files in the Pages subdirectory to create indexes and entries in HTML.

mkcommon.pl holds fixed directories and configuration file names (areas.data, authors .data, and types.data), and defined fields. It also has utility functions, such as concatenating files, rewriting LaTeX-like expressions and cross references, reading configuration files into data structures, and reading term files into data structures. It is broken out so other programs, like generating ontologies, may share the routines and information.

mkcommon.pl uses the file latex2html.data to convert LaTeX-like commands to the appropriate HTML for output. It compiles each replacement in latex2html.data into replacement patterns for simpler processing later.

For each term file one or more items are inserted in the `entry` hash-of-hashes. (@AKA and @WEB fields generate additional entries.)

The code uses several different versions of the entry name for different purposes. The versions, listed by variable name, are

NAME (ename, in other places) original name, e.g., `$B<sub>k</sub>$ tree`, `$\Omega$`, or `K&ouml;nigsberg bridges`

XNAME name without LaTeX markers to look up cross references, e.g., `B<sub>k</sub> tree`

DNAME HTML display name, e.g., `BB(&alpha;) tree`

TNAME plain-text name, that is, without HTML or LaTeX markers, for text contexts like page title, e.g., `Bk tree` or `Omega`

alphaName name used to alphabetize, e.g., TWOTHREETREE or ␣OMEGA (where ␣ is a blank space)

For @AKA and @WEB entries, an HTML file name is created from the name with similar processing.

About half of the code of mkterms is in compiling entries, Step IV in the above list. Most of that code, one-third of the total, writes the entry HTML page. If the entry page is out of date or the "write all" command line option is used, the entry page is started. Otherwise processing proceeds to the next entry.

The area, the type, and several versions of the name are retrieved, and the entry page introduction is copied, replacing certain variables with the retrieved information. The code then has roughly 16 chunks, one for each possible field. Here is pseudocode showing the basic flow for each chunk. (Some fields may not have all processing shown.)

```
if (any material was given for FIELD) {
    retrieve material.
    format material for HTML, which may be to
    rewrite cross references
    rewrite LaTeX expressions
    other formatting
    write out material with HTML labels
}
```

Finally the entry page conclusion is copied, replacing variables.

4.3.1 Minimum Number of Characters

To minimize the number of characters needed to access the dictionary, we made a two-level index, which is at https://www.nist.gov/dads/terms2.html. The first page has a list of second-level pages, each of which links to entries.

Let e be the total number of entries in the Dictionary. Roughly speaking the first page has links to $\sqrt{e}$ second-level pages, each of which has links to $e/\sqrt{e} = \sqrt{e}$ entries for a total download of about $2\sqrt{e}$ instead of e. Exactly how many entries should each index page have for the minimum total number of characters?

We define the following variables:

e - the number of entries

n - the number of second-level index pages

The number of entries per index page is then e/n.

H_m - HTML overhead for main page (the intro and conclusion) (characters)

H_o - HTML overhead for each index page (characters)

H_i - HTML href for each index page (characters)

H_e - average HTML href for each entry (characters)

The size of the main page (in characters) is $H_m + nH_i$. The average size of an index page is $H_o + (e/n)H_e$. The total size of the download, D, is $H_m + nH_i + H_o + (e/n)H_e$.

Minimize the download size by setting the derivative of D with respect to n equal to 0 and solving for n.

$$\frac{dD}{dn} = H_i - \frac{eH_e}{n^2}$$

The minimum download is when $n = \sqrt{eH_e/H_i}$. Note that the (average) number of characters downloaded is $H_m + H_o + 2\sqrt{eH_eH_i}$.

We need to make one more design decision. Since the entries are in a subdirectory, the index pages could be in the same directory as the main page, that is, the main page has

```
<a href="t02.html">am - av</a>
```

and an index page, e.g., t02.html, has

```
<a href="HTML/admissible.html">admissible vertex</a>
```

or the index pages could be in the subdirectory. That is, the main page entry has

```
<a href="HTML/t02.html">am - av</a>
```

and an index page has

```
<a href="admissible.html">admissible vertex</a>
```

As of 2013, these are the measurements: e = 1278 entries, H_e = 77 559/1278 = 56 characters (which includes 5 characters for `HTML/`), and H_i = 30 characters (without `HTML/`).

The characters downloaded is minimized by the first scheme: keeping the references to the index pages as short as possible. Computing we get $n \approx 51$ index pages with about 25 entries per page. That comes out to be about 6000 total characters downloaded as opposed to 89 000 through the main page.

Currently mkterms computes the optimal number of entries per second-level index page (optimEntriesPerPage) and starts a new page when that number is reached. For ease of reading, slightly more or fewer entries should be included in index pages so breaks are at higher levels, or, shorter strings. More thoughts on this are in Sec. 5.2.2.

4.4 Checkers and Helper Programs

4.4.1 Automated Regression Test for mkterms - aaaaTestScript

We have two test cases for the term compiler, mkterms. The first is `aaaaTest.trm`, which has every legal construct and strange case we could imagine. It is for debugging and regression testing, especially rendering math expressions and doing cross references. We

have not checked its coverage on the code. While adding capabilities, we used a test-driven development and added a test for a new capability first.

Accompanying the test file, there is the script aaaaTestScript, which uses expected output in aaaaTestDir. To use it, copy aaaaTest.trm to the Term subdirectory, run mktest (`$ make site`), then run aaaaTestScript. The script checks that the proper files were created with the expected content.

Remember, before publishing DADS, remove the test files created. Compiling aaaaTest.trm creates five HTML files in the target directory, plus entries in the indexes.

The second test file is zzzzTest.trm, which has every error we check for: missing @NAME, misspelled fields, unknown authors, etc. It tests the checking and auditing functions. Copying this to Term and running mktest will report many errors and checks. The file itself contains documentation of many of the expected errors.

4.4.2 Check for Invalid URLs

DADS has links to almost 800 outside web sources. With this many links some of them move or disappear every week. It is not feasible to check these links manually. An automatic link checker is necessary. Checklink checks all the links every Monday morning. Documentation is at https://metacpan.org/pod/distribution/W3C-LinkChecker/bin/checklink.pod. Here is the meaning of the switches we use and the actual command. (The actual command is broken into lines arbitrarily here.)

```
# --summary   don't show details of link checking
# --location http://www.nist.gov/dads   follow docs under this
# --location http://xlinux.nist.gov/dads  ... and under this, too
# --masquerade "http://xlinux.nist.gov/dads// http://xlinux.nist.gov/dads/"
#        allow for NIST redirect mess
# --suppress-redirect 'http://www.nist.gov/->http://www.nist.gov/index.html'
#        don't report this
# --exclude-docs dads/termImpl.html  skip links in this
# http://xlinux.nist.gov/dads/  start here
checklink --summary --location http://www.nist.gov/dads/ --location
http://xlinux.nist.gov/dads/  --masquerade "http://xlinux.nist.gov/dads//
http://xlinux.nist.gov/dads/"   --suppress-redirect 'http://www.nist.
gov/->http://www.nist.gov/index.html' --exclude-docs dads/termImpl.html
http://xlinux.nist.gov/dads/ | bin/checklFilt | sort | bin/checklUnfilt
```

We filter the output to remove lines and whole sections we don't need, then sort the warnings by entry to keep them in the same order each time. Since some resources return error messages to robot scans, like this scan, but are otherwise fine, we compare the output with the output from last week using the Unix diff command. The output from diff is emailed to the author to check.

The files used in this link checking have the prefix .checklink. so they don't show up in typical directory listings.

Most (more than 95 %) of the problems, like page not found, are transient and go away. We ignore most "server not found" and many "page not found" reports in hopes they come back later. However, incorrectly disregarding a problem allows some permanent errors to disappear from the report, since they are the same from week to week. Every other month or so we compare the output to a master output, reference.checklink, to discover such masked problems. We update the master output manually to include new results that are fine.

4.4.3 Updating the External Web Site - publishDads

The mechanism for getting the DADS web pages to the outside server has changed many times. We encapsulate the steps in the script publishDads. In the past the script only moved files that changed since the latest publication. This code and code for transferring files by ftp is in the revision control system.

The current script copies all files to the server using scp. Note that it does not remove deleted files; this must be noticed and done manually. Also the script does not handle files with a space in the name.

The script needs no arguments. It is invoked as

```
$ ./publishDads
```

The scp command asks for the user's password on the destination machine.

4.4.4 Other Checkers

Makefile encodes other quality checks. A spell check rules takes all the source pages, including term files and .intro or .concl files, removes HTML and LaTeX mark-up and URLs, puts every word on its own line, runs a spell checker, then compares the output with the master spell check file, dads.spell. This shows only the new words that the spell checker does not recognize.

Switching to a new spell check program (or none at all!) is easy. Run the spell check and fix anything to get a known-good result. Switch to the new spell checking and rerun the spell check. Whatever results is known to be good and may be used as the master spell check file from now on.

In the past, code errors have allowed incorrect HTML to appear in the final files. To minimize the chance of this in the future, we added a rule to report what may be malformed HTML. It looks for the following:

- backslash (\), caret (^), or dollar sign ($) - these are likely from handling LaTeX-like commands incorrectly
- underscore (_) outside paired double quotes (") - again this is likely to be incorrectly handled LaTeX-like commands. Since URLs often have underscores, we exclude underscores that are within a pair of double quotes.
- a pair of quotes (') - likely incorrect conversion from ``word'' to "word" or unmatched quotes.

- a pair of periods (.) - likely incorrect handling of URLs or an incorrect URL.
- ampersand (&) not followed by a letter
- dangling hrefs - unknown cross reference in an entry

Any findings are compared against a master HTML warning file, htmlWarnings, to show instances not known to be correct.

When preparing entries for the printed dictionary, we generated LaTeX instead of HTML. There is a rule to check for likely errors in the LaTeX output.

The Makefile also has rules to find term files without a definition and new or modified term files.

4.4.5 Editing Many Terms - editTerms

This program is used to perform the same edit on many files. For instance, when the URL for a widely referenced web site changes, update HTML tags to conform to Extensible HyperText Markup Language (XHTML), or change a common word ("vertexes" instead of "vertices").

To use it, change the program so it has the edit hard-coded. Then run it on the files that need to be edited, for instance

```
$ ./editTerms ‘grep -l vertices Terms/*.trm‘
```

The program does the following:

1. edit the file, including changing the *modified line
2. show the differences, so user can check that nothing was unintentionally changed, and
3. prompt for commit (check in: ci) or other action.

Upon seeing the differences, the user has the following choices:

Yes check in

Leave leave edits as is, but don't check in

No unlock file and revert to previous version

Quit unlock file, revert to previous version, then exit

We added L, N, and Q in case the hardcoded edits didn't go as wanted for some reason. It is easier to check the changes to a hand full of files than to test and verify that the code will *always* do exactly what is wanted in all cases.

4.4.6 Renaming Terms - mvterm

mvterm does many tasks needed to change the name of a term file:

- rename the term file, and
- remove all web pages referring to the term file, so that the web pages will be regenerated with the right URL

Note: mvterm does not remove the web page from the external server. That must be done manually.

Why is such a program needed? Occasionally entries are renamed, like changing `perfect shuffle` to be `ideal random shuffle` or `edit distance` to be `Levenshtein distance`. (See those terms for the reasons for the name changes.)

Another use arises because originally we decided that term file names should be no more than 12 characters long. (We don't recall why 12—maybe because of old operating system file system limit?) So there are dozens of files with names like alterntngpth.trm (`alternating path`), communseqpro.trm (`Communicating Sequential Processes`), and nPcompltlngg.trm (`NP-complete language`).

4.4.7 Making Copies to Share - makeTermTar and makeDownloadTar

People have requested local copies of DADS. We are happy to share what we can. Entries from the "Algorithms and Theory of Computation Handbook" (CRC Press LLC, 1999) are copyright by CRC Press, which only gave us permission to put their entries on our web site. Both scripts delete entries that are copyright CRC Press.

The script makeTermTar creates a tar file of all the term files.

makeDownloadTar creates a tar file of the whole DADS web site.

4.4.8 Write LaTeX Version - dumpterms

For the final printed dictionary the entries needed to be in LaTeX format. We wrote a special program, dumpterms, to compile the term files to the proper format. We copied what was mkterms at the time and changed the needed code. It has not been updated since then. It uses html2latex.data.

5. The Future

This section has thoughts of what changes and improvements might come to DADS, and possible approaches or concerns.

5.1 Elaborating Entries

Initially term files only had a few fields. We added fields to better label the information. In this subsection we list even more fields that we might add and other changes we could make.

- Pre- and postconditions are information for reasoning about programs. Adding pre- and postconditions (or any new fields, like @OPS or @EXP or @COMPLEX below) are straight forward. Define new fields, say,

  ```
  @PRECOND=
  @POSTCOND=
  ```

  Capture them in a few data structures in mkcommon.pl and mkterms (Sec. 4.3) and come up with HTML rendering. One thing to decide is whether to allow automatic cross references, like {list}, or LaTeX-like commands. It may also be good to define some (semi-)formal language in which specify pre- and posts, instead of English. Perhaps pre- and postconditions should be in the same field, e.g.

  ```
  @PREPOST=Precondition is ... Postcondition is ...
  ```

- Add @OPS, the basic operations of an abstract data type or data structure. For instance, operations for `dictionary` are dictionary.new, dictionary.insert(k, v, D), dictionary.find(v, D), and dictionary.delete(k, D).

- Add separate fields for complexity, e.g., @COMPLX. Any such field has to support structure. For instance, data structures often have many operations, hence many complexities.

  ```
  linked list
   find item Theta(n)
   insert at head Theta(1)
   insert after item Theta(1)
   insert before item Theta(n) (find predecessor, insert after)
   delete next item Theta(1)
   delete current item Theta(n) (find predecessor, delete next)
  ```

  `Tree`, for example, has both time and space complexities. The `Quicksort` algorithm has both average and worst case complexities. Some algorithms, such as string algorithms like `Karp-Rabin`, `Knuth-Morris-Pratt algorithm`, or `Boyer-Moore`, have significant pre-processing time, too.

  Complexities should be printed when an algorithm is in the index-by-area page, index-by-type page, or the entries-with-implementations page. Maybe there should be a new page that lists all entries, either data structures or algorithms, that have complexities.

- Add a separate field for SOLVED BY. For example, the problem of matching in a weighted, `bipartite graphic` is solved by `Munkres' assignment algorithm`. Symmetry suggests there should be a SOLVES field, too.

- Consider creating a new field, like @IMA and @VARIANT, for "opposites". Consider creating a new field for "may be implemented with". Consider creating a new field for "similar". The IEEE Standards Style Manual 10.4.2 says

  > Subdefinitions of a term shall be marked as (A), (B), etc. ... Contrast: refers to a term with an opposite or substantially different meaning. Syn: refers to a synonymous term. See also: refers to a related term. See: refers to a term where the desired definition may be found. The cross-references listed under those headings shall be in alphabetical order and separated by semicolons ...

- Add @EXAMPLE. Entries like `memoize`, `repeated squaring`, `post-` and `preorder traversal` have embedded examples. It may be helpful to label them explicitly rather than dumping them in @NOTES.

- Replace the current use of # in cross references, Sec. 3.4.1, with | to match Wikipedia markup. A simple script to change term files and a little testing should handle this change.

5.2 DADS Site Enhancements

This section has ideas to improve the DADS site overall, instead of changes to individual entries.

Have a page (maybe the main page) that only lists major entries, however that's rated. It is likely that the top half of entries have over 95 % of the accesses.

Move the bibliography, which is now on the main page, to a separate page. Automatically compile references, e.g., [Knuth98].

Create a history page. Tell about the origin of the dictionary, "philosophy," and other stuff including some notes in this document.

Create indexes for historical and bibliographical entries, like those for types and areas.

Move some entries to Wikipedia, or add links to entries where appropriate.

AlgoVista was a great site that allowed one to look up an algorithm by its function, that is, by what it did. Unfortunately, the web site has been on and off the web over the years. It would be great to resurrect the work and put it on-line beside DADS. It was hosted at the University of Arizona. A Google search leads to a few papers on it.

While checking links, watch for "moved" notices as well as missing. Notices that a page has moved might not have a link, so the information is gone and the link should be updated. Even if there is a link, a moved notice doesn't last forever. We should be aware of it and update the reference quickly. A move notice may be indicated by the words "relocated" or "update" and "link(s)" and a short page.

Change mkterms so that the meta tag in the header doesn't have "Defin. of ..." if there's no definition. Also adapt "with possible links to" to be context sensitive, that is, say "links to" if it has links, but otherwise, nothing.

Some spell checkers allow for a user-specific dictionary. We could maintain a dictionary specific to DADS for the spell check, rather than filtering incorrect words afterward.

5.2.1 Future of Page Compiling

Currently DADS is compiled into static web pages by custom software, mkterms, as explained in Sec. 4.3. Instead of making all the changes mentioned in this section to our own software and supporting it, it may be useful to switch to a widely used static page compiler, such as a more modern version of Octopress, Jekyll, or Pandoc. Octopress supports web features such as

- HTML5,
- layouts responsive to mobile formats (rotate and resize), and
- 3rd party social networks, like Twitter, Pinterest, and Delicious.

5.2.2 Automatic Optimal Two-Level Index Breaks

Currently entries in the two-level index are divided so an arithmetic optimal number of entries is in each index page, as described in Sec. 4.3.1. Some of the indexes generated by these arithmetic breaks are `array search`—`balanced quicksort`, `balanced tree`—`binary tree`, `binary tree representation of trees`—`bottleneck traveling salesman`, `octree`—`ordered binary decision diagram`, `ordered linked list`—`partially ordered set`, and `partially persistent data structure`—`pipelined divide and conquer`.

For ease of reading, entries should be broken to trade off an optimal number of entries per index page with breaking at shorter prefixes, like volumes of some encyclopedias. For instance,

- α—ah
- al—ax
- b—bin

The algorithm to find breaks needs to minimize some fitness function that gives more weight to breaking between short strings (between ny and O, instead of between `occurrence` and `octree`) and that also gives weight to getting the optimal number of entries per index page. Coming up with a fitness function is difficult. Minimizing the fitness function is hard, too (probably not P). Dynamic programming is one approach. Another is a heuristic, like the "blocktimize" algorithm.

An excellent paper on developing a linear time paragraph formatting (breaking) algorithm is J. Gibbons and O. de Moor, *Bridging the algorithm gap: a linear-time functional program for paragraph formatting*, Science of Computer Programming, Vol. 35, No. 1, September 1999.

5.3 Supporting Languages Other Than English

It is not clear how best to incorporate other languages in DADS. Here are some possibilities. All have flaws, so we hope someone can come up with something better.

There are three considerations: how to place other material in term files, how to display the information, and how to change all the coding for dozens of hard-coded English strings in the software.

5.3.1 Other Languages in Term Files

One big concern is setting up the format to increase the chance that if the English entry is changed or links added, entries in other languages are kept up to date.

Let's consider just two fields, name and definition, for one entry, `linked list`, which is in linkedList.trm. Currently the fields are

```
@NAME=linked list
@DEFN=A {list} implemented by each item having a {link}
to the next item.
```

For reference, here are Google translations into Dutch

name: verbonden lijst

definition: Een lijst die door elk item met een link naar het volgende item.

We could have everything for other languages in a different file with some special attribute. For instance, put material in linkedList,DUTCH.trm But someone might fix a definition or add a link in the English file and forget to update other languages. It seems like it would be best to keep everything in the same file.

We could have the same field identifiers, but with a special attribute

```
@NAME/DUTCH=verbonden lijst
@DEFN/DUTCH=Een {lijst} die door elk item met een {link}
naar het volgende item.
```

What if we don't have "list" (lijst) in Dutch? Maybe all parenthesized links should be to the English entry, so

```
@DEFN/DUTCH=Een {list} die door ...
```

We can't require that every entry has every language, so each language entry should be tagged somehow.

Maybe we could handle irregular plurals with something like

```
de {children#child|DUTCH:kinderen#kind} van elke knoop zijn
```

Another problem with links in other languages is smart linking. Currently if a cross reference doesn't match anything, a trailing "s" or "es" is removed or a leading upper case letter is changed to lower case to find the reference. Should automatic plural handling be added for other languages?

Maybe instead we should handle grammatical work variations in other languages like English (with the # construct) and just have an indication of the English entry

```
de {kinderen#kind|child} van elke knoop zijn
```

The above would be

```
@DEFN/DUTCH=Een {lijst|list} die door ...
```

By the way, *great* care must be taken in choosing a special character, like # or $\{\ldots\}$, in syntax. There are only so many characters, and nesting them can get very messy.

Maybe each definition can carry a date. If another language date is older than English, "This may be out of date" (in the appropriate language) is automatically added.

5.3.2 Displaying Other Languages

DADS will not have exactly the same material in all languages. Some languages will have some links or definitions or entire entries that other languages don't have. How should different languages be displayed?

We could interleave them. (The following approximates what the user sees in their browser.)

```
linked list
verbonden lijst

Definition: A list implemented by each item having a link to the next
item.
Een lijst die door elk item met een link naar het volgende item.
```

We could generate a complete Dutch (or other language) version of the entire DADS. English web pages are used whenever there is nothing in the other language. So when one accesses, say

```
www.fastar.org/DADS/DUTCH/verbondenLijst.html
```

one sees

```
verbonden lijst

Definitie: Een lijst die door elk item met een link naar het volgende
item.
```

```
Noot: The first item, or head, is accessed from a fixed location,
called a "head pointer." An ordinary linked list must be searched with
a linear search. Average search time may be ...
```

More information (@LINKS) should definitely have some marking for links in other languages, maybe like this

```
More information

an introduction, a Java applet animation (Java).

(Dutch)
borrelen soort een verbonden lijst (link to
http://faq.programming4.us/nl/software/98386.aspx)
```

Maybe Implementation should have separate sections, too? Or if we have separate sites (an English subdirectory and a Dutch subdirectory), maybe the other sites should add English @LINKS and @IMPL by default *unless* a flag says, "don't show these in the Dutch linked list page."

Maybe a better idea is to render everything we have in Dutch, then put *all* the English material below. That way the reader may decide if anything is out of date or decide to follow English links. For instance,

```
verbonden lijst

Definitie: Een lijst die door elk item met een link naar het volgende
item.


linked list

(data structure)

Definition: A list implemented by each item having a link to the next
item.

Also known as singly linked list.

Specialization (... is a kind of me.)
doubly linked list, ordered linked list, circular list.

See also move-to-front heuristic, skip list, sort algorithms: radix
sort, strand sort.
```

```
Note: The first item, or head, is accessed from a fixed location,
called a "head pointer." An ordinary linked list must be searched ...
```

But if one follows a link to an English entry, how would one get back to Dutch entries? Ah, but if the entire dictionary is rendered for Dutch, all the links stay within that dictionary so it would link to Dutch entries when they are available.

Vreda Pieterse suggests [6] that structures defined in a topic map standard might be the way to go, i.e., providing an optional scoped entry for anything—fields, links, etc. If the site is compiled for example with some flag to compile the DUTCH version of DADS—everything that contains a DUTCH scope will then be rendered with the info in that scoped section, while if it does not have a DUTCH scope, the default global scope (or parent scope) will be rendered.

5.3.3 English Infrastructure

One of the tasks to support other languages is to remove all hard-coded English from the code and put such strings in a special file. For instance mkterms has code like the following.

```
"$dname: see <a href=\"$entrypg\">...
(no definition here, yet, but <a href=\"../$MAINPAGE\#needHelp\">you
can help</a>.)
Formal Definition:</strong>
Generalization</strong> (I am a kind of .
```

The GNU Native Language Support (NLS) project uses an interesting approach. It is documented in info gettext, and uses ISO 639 country codes. It replaces each string with a number, then a language support file has entries in that human language, like

```
see
no definition here, yet, but
you can help
Formal Definition
Generalization
I am a kind of
```

To support a new language, one may copy the English language file, then change whatever one wishes.

At one time there was a Persian and a Catalan translation of DADS. These could be merged into the main DADS.

5.4 Other Formats

5.4.1 Computer Access

Currently DADS is only accessed through HTML. We have had requests to make it available in ontology, Z39.50, DICT, and Extensible Markup Language (XML) formats. These shouldn't be too hard, but we haven't gotten around to it.

5.4.2 Representation of Mathematical Expressions

Currently DADS uses an ad-hoc LaTeX-like format to represent mathematical expressions. The format is explained in Sec. 3.4.2. Twenty years ago that may have been reasonable, but there are widely used alternatives now. DADS should use MathML or perhaps Open Math.

MathML is in development. Details are available at https://www.w3.org/Math/. It is distinct from LaTeX, but converters are available. Definitions should be written in MathML (or move that direction), and software in mkterms convert to HTML.

Open Math is an emerging standard to represent the *semantics* of mathematical expressions [7], whereas MathML deals with their *presentation*.

References

[1] Gaede V, Günther O (1998) Multidimensional access methods. *ACM Computing Surveys* 30(2):170–231. https://doi.org/10.1145/280277.280279

[2] Miller GA (1990) Nouns in WordNet: A lexical inheritance system. *International Journal of Lexicography* 3(4):245–264. https://doi.org/10.1093/ijl/3.4.245

[3] Black PE (2002) Personal communication.

[4] Howe D *Instructions for FOLDOC Guest Editors* http://foldoc.org/editing.html.

[5] Leuf B, Cunningham W (2001) *The Wiki Way: Quick Collaboration on the Web* (Addison-Wesley).

[6] Pieterse V (2016) *Topic Maps for Specifying Algorithm Taxonomies: a case Study using Transitive Closure Algorithms*. Ph.D. thesis. University of Pretoria. Available at http://hdl.handle.net/2263/59307.

[7] *OpenMath* http://www.openmath.org/.

A. An Example Term File

```
# *created  "Tue Mar 16 16:31:33 1999" *by "Paul E. Black"
# *modified "Thu Aug 14 12:14:40 2008" *by "Paul E. Black"

# $Log: invertedIndex.trm,v $
# Revision 1.4  2008/08/14 16:14:41  black
# Update HTML line break (br) tags to be XML/XHMTL compatible
#
# Revision 1.3  2006/10/13 17:18:42  black
# move appropriate XREFS to VARIANT.  Add BIB for Zobel & Moffat 2006.
# NOTE that word number, weights, etc. may be in the index.
#
# Revision 1.2  2004/12/17 16:52:33  black
# Refine XREFS into IMA, etc.  Add RCS keywords.  Make more XHTML compliant.
#

# entry name
@NAME=inverted index
# _A_lgorithm, _D_efinition, _P_roblem, or data _S_tructure
@TYPE=S
# basic theory numeric search sort tree graph combin(atorial) para(llel)
@AREA=search
# the definition
@DEFN=An index into a set of {texts} of the words in the texts.
The index is accessed by some {search} method.
Each index entry gives the
word and a list of texts, possibly with locations
within the text, where the word occurs.
# formal definition or {cross reference} to an entry
@FORML=
# comma-sep list of pure names or {cross refs} that this is Also Known As.
@AKA=
# other cross-listings solely for the web, such as word or spelling variants
@WEB=inverted file
# comma-separated list of {cross references}, i.e., See also ...
@XREFS={full inverted index}, {inverted file index}, {block addressing
index}, {index file}, {external index}, {forward index}
# bib refs, eg, to defining article (pure HTML).  printed within <p>..</p>
@BIB=<strong>Nivio Ziviani, Edleno Silva de Moura, Gonzalo Navarro,
Ricardo Baeza-Yates</strong>, <em>Compression: A Key for
Next-Generation Text Retrieval Systems</em>, IEEE Computer,
33(11):37-44, November 2000, (page 42).
```

```
# any notes.  these will not be printed in the final dictionary
@NOTES=Suppose we want to search the texts "i love you," "god is
love," "love is blind," and "blind justice." (The words of the text
are all lower case for simplicity.)  If we index by (text, character
within the text), the index with location in text is:
<pre>
blind   (3,8);(4,0)<br>
god     (2,0)<br>
i       (1,0)<br>
is      (2,4);(3,5)<br>
justice (4,6)<br>
love    (1,2);(2,7);(3,0)<br>
you     (1,7)
</pre>
The word "blind" is in document 3 ("love is blind") starting at
character 8, so has an entry <code>(3,8)</code>.
To find, for instance, documents with both "is" and "love," first
look up the words in the index, then find the
intersection of the texts in each list.  In this case, documents 2
and 3 have both words.  We can quickly find documents where the words
appear close to each other by comparing the character within the
text.
# further explanation (pure HTML)
@LINKS=
# implementation(s) (pure HTML)
@IMPL=
# author's initials
@AUTHOR=PEB
# end $Source: /home/black/DADS/Terms/RCS/invertedIndex.trm,v $
```